FN-383
2380.000

FERMILAB'S ADVANCED COMPUTER R & D PROGRAM[*]

Thomas Nash, Stephen Bracker[†], and Irwin Gaines

April 1983

[*]Invited paper to be published in the Proceedings of Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.
[†]Bracker Research, Ltd., Toronto, Canada; on leave from the University of Toronto.

FERMILAB'S ADVANCED COMPUTER R & D PROGRAM

Thomas Nash, Stephen Bracker[+] and Irwin Gaines
Fermi National Accelerator Laboratory
Batavia, IL 60510 USA

By any standards the analysis computing requirements of Tevatron fixed target and colliding beam experiments at Fermilab will be huge. Table I shows some early predictions[1]; many of these are likely underestimates because they assume large improvements over existing algorithms. This experimental load will have to share facilities with lattice gauge calculations, a very important theoretical subject stirring much interest at Fermilab.

Fermilab's response to this problem is on two fronts. A "Request for Proposals" has been released for a $5 million acquisition (by lease) intended to at least double the present capacity of the Computer Center's three CDC Cyber 175s. The schedule of this competition projects to a delivery at the end of 1983.

The second response to the computing problem is based on the recognition that commercial solutions do not appear likely to meet the full requirements of high energy physics over the next decade. An effort aimed at confronting these computing bound problems of high energy physics has been organized at Fermilab and named the Advanced Computer R & D Program (ACP). The intention is to create a stimulating atmosphere in computer research and development in which new approaches to computer design will flourish. Interactions outside high energy physics with computer scientists from industry and universities are strongly encouraged. Particularly successful has been a seminar series held over the past year. This has brought to Fermilab a very distinguished list of specialists (see Table II) covering many diverse aspects of computer science activities in the U.S. There has been a very useful two way communication both during and after these visits.

We hope that the ACP will ultimately act as an umbrella program covering several simultaneous projects. During the first period, however, there will be a single focus. As an R & D effort, the program has no immediate operational responsibilities. Therefore, though well supported, it has intentionally been made organizationally independent of Fermilab's Computer Department. On the other hand, the computing requirements for Tevatron experiments cited earlier make clear what direction the first efforts should take, namely the development of a reconstruction processor. Building on the foundation of personnel, equipment and module designs established by the first project will be, we expect, a subsequent effort aimed at the lattice gauge problem. Typical of the type of research we would like to pursue in the future is work on large data base technology allowing for fast analysis histogramming at convenient work stations.

In the rest of this paper, we will describe the ongoing work on the reconstruction processor project much of which presently is focused on the conceptual design. Previous experience with several earlier processor systems by people associated with the project is influencing present thinking. Two of these earlier systems were developed at Fermilab and have been described in detail elsewhere[2]. The M7 Processor was designed in 1977 by T. Droege, I. Gaines, D. Harding and K. Turner for use in triggers on experiments in the broad band photon beam. This is a fast cycle (110 nsec), streamlined, stored program computer with a specialized instruction set. These microprogrammed instructions are of the form $E_k = A_i * B \pm C_j * D$ (where the $\pm$ may be any ALU operation). This was based on the very important understanding that (at least for most fixed target experiments) the innermost loops of track reconstruction programs are dominated by linear operations of that kind. Although originally intended for triggers and not FORTRAN programmed, the M7 is now being prepared for interspill and offline computation.

The ECL-CAMAC Trigger Processor System, designed in 1978 principally by E. Barsotti, S. Bracker and T. Nash, is modular and data driven and capable of extremely high speed computations involving loops and subroutines. It is programmed primarily by the way modules are interconnected - though FORTRAN is used to program memory (table) look up units. Decidedly non-Von Neumann operations lead to tremendous operational cost effectiveness. These include a 120 ns per loop line finder that uses projections to a bit list and the table look ups. An algorithm that required an average 7 μsec on a $100K trigger system needed ~40 msec on a Cyber 175. This power comes at the cost of inflexibility and difficulty in programming.

There has also been some experience with two important systems developed outside of Fermilab but in use for Fermilab experiments. These are the Nevis Data Driven Trigger Processor (designed by B. Knapp and W. Sippach) and the SLAC 168E emulator (P. Kunz).

Our basic goal in designing a new reconstruction processor is to obtain as much as possible of the power of the data driven processors in a system at least as programmable and flexible as the emulators. At first, to meet the fundamental needs of the Tevatron experiments we need to reach a cost effectiveness (units of equivalent megaflops/mega $ are appropriate) 100x that of the Cyber 175. This recognizes that such a system can not replace the services of a computer center. Thus typically amounts of money representing only 10% of the computer center value will be available to fund a system - aimed to be 10x the computer center processing capability - to handle the most time consuming problems. Such a system must be comfortable for physicists to use and program and be flexible. This means a very friendly host system with excellent software development tools either on the host or at the computer center. After much debate we are convinced that we must provide FORTRAN (with extensions likely) as the first priority language. Some other more modern alternative languages should also be available.

We have just described what are essentially minimum goals. Regrettably, for an R & D program, these are almost operational in nature. Our real goals - and interest - are to get as much processing power as we can (another 10x-100x) and still have a friendly, programmable system. Why provide more than "needed"? First of all, we are convinced that the present projections of needs are, as is usually the case, underestimates. The minimum goals will, we suspect, barely meet demands and then only in a near saturated environment. The real motivation is to remove computing restrictions as much as possible from physics choices. The type of experiment and detector chosen or a decision to rerun a major reconstruction for better resolution, for example, should be decisions based on physics considerations not computer turn around time. Most importantly, human, not computer, limitations should dominate the time required to iterate physics. Physicists spend far too much of their time shepherding production jobs or waiting on computers to turn around physics ideas. Major improvement in this area will affect directly the productivity of physicists and physics installation.

Our design approach is to combine the power of specialized devices (like those used in data driven triggers) with the programmability of 16/32 bit microprocessors or small computers that are supported with good FORTRAN compilers. The system will be very modular in concept to allow for optimizing architecture - maximizing hardware utilization - for different classes of (and specific) problems. This includes the possibility of an array interconnection of processor nodes for lattice gauge calculations[3] as well the stream independent multiprocessing approach natural for event reconstruction which we will describe below. Modularity is also of great value in allowing systematic debugging of complex systems. The planned 3081E processors with a large improvement in modularity over their 168E predecessors demonstrates that this lesson has been learned in the high energy physics community[4].

Our staged plans for developing a powerful multiprocessing system are encouraged by four important characteristics of events reconstruction algorithms. Four levels of system concepts with increasing complexity are indicated in Figures 1-3. Each demonstrates one of the characteristics:

Events are independent of each other. This allows large numbers of small, highly cost effective processor nodes to compute on single events independently - without the complexity of inter node communication (Figure 1). Event processing time is long enough and event lengths short enough so that bus communication at the input and output will not be a practical problem for this type of system. We estimate the limitation to be upwards of 100 nodes for typical event reconstruction with bus loads comfortably within the capability of Fastbus but near the maximum data rate of a single 6250 bpi tape drive.

The problem breaks easily into major serial subroutines. These subroutines, usually written by different groups of people, carry out the reconstruction of different detectors or classes of data. This naturally suggests the Level 2 structure of several ranks of nodes (Figure 2). The number of nodes in each rank is adjusted to balance the system for a particular program and avoid bottlenecks. Events streaming from rank to rank are assigned to any available node. Arbitration is trivial in the switches which are really busses since the throughput is such that only one event need be handled at a time and there are no constraints on the choice of node since each has the same program load and hardware.

There exists a kernel of basic instruction sequences that dominate the computing time. This means that special purpose devices - which we call coprocessors - may be brought to bear effectively on the computing intensive kernels of the reconstruction algorithm. These coprocessors are more powerful than what is usually meant by the word when referring to floating point coprocessors. In fact they are special purpose hardware that will often use the non-Von Neumann techniques such as projections to hit arrays of the data driven triggers to find, match and fit tracks. Special SU(3) multiplier coprocessors will be very useful for the lattice gauge problem[3]. Operating with very high cost effectiveness, they will take control from the primary processor on an extended FORTRAN instruction similar to a subroutine call for a restricted list of names, such as CALL LINE

(...,...,...). These "hardware subroutines" will be documented well and their use, from a programming point of view, will be similar to calling a CERNLIB subroutine. The multiple ranks of the Level 2 concepts have their primary advantage when coprocessors are added (Level 3, Figure 2). Coprocessors of a particular type are only installed on ranks where they are required. The multiple ranks allows optimizing hardware utilization of the coprocessors. We will return to a more detailed discussion of coprocessors later.

There is intrinsic parallelism within an event. An example of this is the possibility of reconstructing line segments in several parts of a detector simultaneously before linking them. Use of such a Level 4 concept (Figure 3) may ultimately be valuable for optimizing coprocessor use. Other sophisticated approaches such as sharing processor/coprocessor teams between two events are also of imaginable benefit but are not shown in Figure 3. These kind of advanced and complex strategies for extracting the last factor of two in cost effectiveness will be avoided for some time in our project because of the desireability of maintaining simplicity in the design.

At this time our primary efforts are aimed at selection of an appropriate processor - compiler combination in the context of the Level 1 concept (Figure 1). This type of system, should with a straight forward design, reach a cost effectiveness ~100 x Cyber 175 in terms of real physics code. As noted earlier the processor cost effectiveness of good microprocessors is very high and memory at appropriate speeds is cheap enough to contain whole programs generously without adversely effecting the cost effectiveness. In fact the large majority of processors we are considering have address spaces larger than the Cybers.

The important factor is turning out to be the quality of FORTRAN compilers that are available. The microprocessor compiler situation is improving rapidly. We are bench marking compiler-processor combinations using a reconstruction package from a typical large Fermilab experiment (E87-400).

There appears to be essentially three catagories of processors in our considerations. These are: a) Easily available, continuously developing families of processors such as the Motorola 68000 and descendents and the Intel 8086 and its descendents. The quality of FORTRAN compilers for such systems is a key issue in our testing. b) Super powerful chip sets associated with available or potential small computers such as the "VAX on a chip" and the HP9000. These are well supported with software but unavailable as chips, apparently because of marketing considerations. A research agreement motivated by some of the novel approaches we are planning to take may hopefully, if appropriate, make such chips available for our research efforts. c) Full small computers, the 3081E for example, may be appropriate for our purposes. However, the scale of multiprocessing would change and it is not clear how much could be gained with coprocessors with such large and expensive primary nodes.

An indication of the comparison criteria we are using is shown in Table III. A large list of possible candidates, not all of which will be under detailed consideration, is in Table IV. We expect that after a somewhat quantitative comparison we will have several reasonably acceptable possibilities. The final selection will be based on some of the more subjective criteria having to do with the host system and the availability of a research agreement.

We have already referred to specialized coprocessors that take control from the primary processors for time consuming inner loops. This is the only approach we are aware of that allows a substantial jump in cost effectiveness beyond the typical level obtainable with microprocessor or emulator multi processing. The figure of merit in using coprocessors, the relative improvement in cost effectiveness is

$$M = \frac{R_C}{(1 - f + f/s)} \qquad (1)$$

Here f is the fraction of time spent in coprocessor code when operating without coprocessors, s is the speedup of coprocessor operations relative to that of the primary processor node alone, and

R_c is the ratio of the cost of a node without to that with coprocessors.

It is clear from this formula that there is an important premium in getting f as large as possible. For s large and $R_c = 1$, the case of high speed cheap coprocessors, $M=\frac{1}{1-f}$. To get an improvement of a factor of 10, $f \geq .9$. This means that the coprocessors' capability must span a very large fraction of the time consuming algorithm kernels. Is it possible to select a class of operations, and design appropriate coprocessors, that have $f > .9$?

To answer this question we have started a detailed study of the structure of reconstruction programs. The results from this study have already proven to be interesting. It appears that the creation and manipulation of lists is by far the most important activity in reconstruction algorithms. It was necessary to develop our own basic notation for such list manipulations. On hearing this described, David Kuck pointed out to us that there is a great similarity in the manipulation operations used in high energy physics to the relational algebra of the computer science research on relational data bases. This very active subject (with its own journal) is considered to be potentially of great importance to many business applications, such as airline reservations, that involve large data bases.

In the following we will describe the list structures and manipulations that appear to cover a very large fraction of reconstruction algorithms. It must be emphasized that our study is still very preliminary. It is based on a detailed examination of the largest code at Fermilab (E516) as well as a less detailed understanding of the algorithms of several other fixed target programs. A planned further study of non-Fermilab and non-fixed target algorithms is not likely to result in a large change in our conclusions, though we can't be certain of this yet. In our description we will use words that are most relevent to the physicist's image of what is going on. These differ from the language and terms of relational data bases which we will indicate in parenthesis using the nomenclature in a manual for a relational data base management system available at Fermilab^5.

A list (relation) consists of some number of named columns (attributes) each containing data (attribute values) of a particular type (Figure 4). Each row of the list contains a set of data such as the hit coordinates of a specific track. There is a unique index for the list which is usually implicit like that of a FORTRAN array though it may be related to a particular column (key attribute).

Starting with a set of independent wire chamber coordinate hit lists as indicated in Figure 5, a reconstruction algorithm works through a sucession of shorter and wider lists (containing track segments, for example) until it obtains the final list containing for each track a row with parameters and the hits that have been identified for the track. At each stage the algorithm is trying to find elements of the lists (such as hits) that it can associate and put into single rows of related elements (such as regional track segments) of the next level of list.

It is important to emphasize that these list structures are not representative of how data is actually stored in memory in programs. Pointers are often used to relate (this word appears frequently) data in one list to that in another rather than creating more and more real lists in memory. These list structures and the manipulations represent conceptually what is going on in the algorithms. An understanding of these conceptual structures of reconstruction algorithms is the first step in defining appropriate specialized hardware coprocessors to carry out these operations in non-Von Neumann architecture.

A limited set of simple and more sophisticated list operations (modification commands and relational algebra operations) is used to build toward the final list. The more simple operations (Figure 6) consists of combining or contracting lists by appending or deleting by rows or columns. (Deleting by columns is called projection in relational algebra). Also shown in Figure 6 is another operation of conceptual importance in building the more complex projection operator to be described below although it does not appear to be used by itself. This is the "product" (X*Y) of two lists created by a double index DO loop over the indices of the two lists.

The most important operation for reconstruction is what we call a projection (not the relational algebra projection). Projections are list operations that relate two lists, X and Y, to create a new list, Z, from selected combinations of rows of X and Y. The selection of rows from X*Y is determined by a projection function. The selection is made by finding those rows of Y for which stated elements fall, within specified ranges, of functions of a single row of X. In the simple projection shown in Figure 7 a single projection is made from X to Y. The nomenclature derives from the common usage where, for example, track segments in one region (X) are projected into a chamber hit list or a track segment list (Y) in another region to produce a track list (Z). In many situations, as when a track segment list is projected into a number of chamber hit lists, several projections are coupled. A predefined number of individual projection functions (say 3 out of 5) is required to be satisfied for the coupled projection to succeed and a new row (track) added to Z (the track list).

The projection function contains much of the diversity required by differing experiment geometries and algorithms. As such, any specialized hardware handling projection functions must be readily programmable. At the same time much emphasis must be placed on optimizing the cost effectiveness and, probably, the speed of projection function hardware since they represent the innermost loops. It appears that, at least in the Fermilab fixed target environment, all projection functions are linear involving sums (or compares) between multiplications. This was the structure first emphasized in the context of the M7 instruction set. At other laboratories in experiments with cylindrical geometries, there appears to be a heavy use of trigonometric functions. We suspect that most of these cases will allow a linearization of the projections. If not, one will have to admit sin and cos operations into projection hardware.

The maximum number of multiplications and additions (or other ALU operations) that we have seen so far to carry out the most complex projection functions in parallel is less than 12 of each. (Detailed cost effectiveness studies that involve balancing the speeds of coprocessors and their processor nodes are required to determine whether, optimally, the projection function operations should be done in parallel or folded through a smaller number of operations.) It is interesting that the SU(3) multiplication that consumes so much time in lattice gauge calculations is made up of 9 * (12 multiplies and 6 adds). Hardware that is optimized for linear projection functions, therefore, looks like it will fit the SU(3) problem in a very natural way.

The study of the structure of reconstruction algorithms allows us to make a preliminary catalogue of the basic coprocessors required. These are:

1. List indexer and element selector. This basic unit controls looping, handles relational algebra involving the conceptual moving of rows and columns, and controls projection indexing.

2. Linear projection function. Called by the coprocessor described above, this device computes projection functions. As noted earlier these functions are primarily linear in form and may be built of a number of parallel multipliers followed by arithmetic logic units (ALUs) to carry out additions and comparison operations. A set of microprogrammed multipliers and ALUs is a natural way to obtain the flexibility and speed so important here. The microprogramming would be compiled directly from user provided FORTRAN projection functions. Folding - that is reducing parallelism by looping results back through the same circuits - may be used to match the cost and speed of this unit to memory and the main processor. If trigonometric functions turn out in fact to be required, they can be added in parallel to the multipliers as look up tables in fast memory or using commercially available

floating point/trig function coprocessors expected to be available soon.

3. Binary search and ordering. Also called by the first coprocessor, this may be required for cases where projections to hit arrays (bit lists) cannot be used because of memory considerations. The most effective way to carry out projections is to a hit array where each location in the array indicates the presence or absence of a hit within the appropriate projection resolution. In this case the projection function simply computes the address in the hit array and looks. For high resolution or long dimensioned projections the amount of memory required for the hit array is prohibitive. It might be possible to break up the projection into a series of increasing resolution steps to reduce the memory required. If such an approach cannot generally be applied in a cost effective manner, it will be necessary to carry out projection searches through whole lists scanning for matches. A coprocessor that does binary searches, the optimal way to do this, is then required. Lists must be ordered for a binary search to succeed and ordering hardware must therefore be provided.

4. Calibrator and list builder. Raw data entering programs must be calibrated and mapped to a standardized format in the special list memory that is used by the coprocessors. In hardware this operation can be described as nothing more than a smart DMA (Direct Memory Access) controller. In some programs the equivalent activity takes 20% or more of the required computer time. Conceptually this unit unpacks data from input common blocks (buffers) putting data into working variables. After processing it transfers results into packed output common blocks (buffers).

5. Fitter. Least squares linear fits are an essential part of all reconstructions both during pattern recognition and in

determining final physics parameters. An interesting and very powerful fitter has been designed by Knapp and Sippach for their data driven trigger system using bit serial modules[6]. We are far from understanding whether such a completely pipelined system can be effectively mated to the coprocessor approach without changing module connections for each use. If possible, however, the Knapp-Sippach fitter would be very desireable because it probably comes close to the limit of cost effectiveness obtainable with a hardware fitter.

It is likely that the list memory accessed by the coprocessors will have to be reasonably fast although final design optimization may point to slow list memory that is identical to the data memory of the primary processors. How big will this possibly fast cache memory have to be for each event? A detailed specification required for the largest active Fermilab code requires a total of 37,600 16 bit words. Even using the most expensive high speed bipolar memory, list memories of up to 64K words will cost under $2000 and appear reasonably matched to a likely scale of costs for primary and coprocessors.

It has been pointed out how important it is for the coprocessors to cover a very large fraction of the time spent in reconstruction processing. We have carefully timed the various parts of the big program used for this first study. In this case the event time is spent as follows:

1.	Setup and calibration.	2. %
2.	List manipulations, projections, etc	74.6%
3.	Fitting	15. %
	Total available to preliminary list of coprocessors	91.6%
4.	Track Selection	7.7%
5.	Miscellaneous	0.7%
		100. %

The potential speedup with our list of coprocessors is therefore ~12x. Track selection involves discarding ghost and other spurious tracks in a systematic fashion using defined criteria. If this activity can be sufficiently generalized to put in coprocessors, the potential speedup becomes still larger. Our plan is to document the preliminary specifications for coprocessors. After circulating them to potential fixed target and colliding beam users, we will learn better what is required in final designs.

We have several times alluded to the need for balancing the speed and cost of the various parts of the system. Equation (1) for the relative increase, M, in cost effectiveness provided by a given coprocessor system is rigorous and a fundamental constraint on design. Design decisions come down to a good understanding of the relationship between R_C, the relative cost with and without coprocessors, and S, the speedup due to the coprocessors. These numbers depend critically on the speed of the primary processor node. For a faster node, it is obviously more expensive to obtain the same speedup, S. But a faster node is itself more expensive and R_C may end up smaller for a given S. In choosing from available design strategies it is therefore clear that one must optimize the overall system figure of merit which is the system cost effectiveness, $C_{system}=M\ C_{node}$. Determining cost versus speed relationships for processors, commercial and custom integrated circuits, and, particularly, for memory is as difficult as it is important. New technologies that open up, the state of the economy, and the politics of foreign trade all have effects on projections on the scale of one and more years in advance that are required for a project like ours.

Several approaches that we have referred to before can be used to optimize hardware utilization. The vertical segmentation into several ranks of processors (Level 2) allows for smaller program and data memories at each node and for more efficient use of coprocessors. Operating within events in parallel on more than one processor/coprocessor system is a sophisticated - and difficult - possible strategy. Far simpler in terms of synchronization is the idea of operating on 2 events simultaneously in one processor/coprocessor system. One event takes precedence and is processed in either the primary or coprocessor at any moment and the other event is used as a filler in the otherwise idling hardware. In ideal situations, though additional memory and registers are required, such an approach can give in principle up to an additional factor of 2 over the basic formula for M of Equation 1. We do not, of course, intend to apply such sophistications until a reliable and stable basic system has been established.

At this stage of the project, we have built up a reasonable and potent repertoire of interesting design strategies. Over the next 6 months the major emphasis will be on measuring and projecting cost versus speed relationships in order to select as best we can, the optimum hardware and design combinations. One aspect of this effort is the ongoing study that will lead to a choice of processor for the primary node and then, as soon as possible, to a first prototype. The other immediate task is to understand projection of memory and VLSI costs for the different available technologies. With the work going on in parallel into coprocessor specification, the project should in half a year be in a good position to produce a powerful and effective system.

Acknowledgements

A number of people were involved in a series of working sessions that established the direction of this project. They include: M. Johnson, D. Kaplan, R. Raja, J. Slaughter and H. Thacker. We have also had useful conversations with A. Brenner. The two new members of the ACP group, H. Areti and M. Fischler, as well as E. Barsotti, have made important contributions in recent discussions. David Bintinger wrote the extremely organized code for E516 that (once he provided comments) allowed us to identify lists and list manipulations as the primary activity of such reconstruction programs. We also thank David Kuck for pointing out the similarity of the list manipulations we identified with those of relational data bases. Finally we thank Leon Lederman for his strong support. His idea of starting a seminar series has proven to be very valuable in establishing communication between the worlds of high energy physics and computer science.

FOOTNOTES AND REFERENCES

+ Bracker Research, Ltd, Toronto, Canada; on leave from Univ. of Toronto

1. Prepared for an early evaluation of the need for a research and development effort in computers, "A Program for Advanced Electronics Projects at Fermilab", Thomas Nash, May 1982.

2. See "A Review of Programmable Systems Associated with Fermilab Experiments," invited paper, T. Nash, proceedings, Topical Conference on the Application of Microprocessors to High-Energy Physics Experiments, CERN, Geneva, 4-6 May 1981. (CERN 81-07) and references contained therein.

3. G.C. Fox, "Construction and Use of a Homogeneous Machine for Scientific Problems"; A. Terrano, "Lattice Gauge Theories and Special Purpose Computers", papers presented to this conference, elsewhere in these proceedings.

4. P. Kunz, "The 3081/E Processor", paper presented to this conference, elsewhere in these proceedings.

5. F.P. Gray and S.O. Wahlstrom, User Guide: RIM 5.0, Boeing Commercial Aircraft Co, Seattle (1982); J. Ingebretsen, Use of RTI/RIM-Test Version, Fermilab (1983).

6. W. Sippach, private communication.

Table I. - Selected Data Intensive Experiments[0]

				Computing Requirements	
Source	Expt.	Year	Data taking rate words/beam clock-sec[1]	$\frac{\text{Cyber sec}}{\text{beam clock-sec}}$	Σ CPU time Cyber years[2]
Butler	E-400	1982	50000	5.0	0.33 (1982 run)
" "	E-687	1986	50000	5.0	1.3
Nash	E-516	1981	10000	1.2	0.75
" "	E-516 seq	1985	10000	1.2	0.5
Jensen, Yoh	CDF	1985	50000	3-6	1-2 (per 3 mon. run)
Cox	E-537	1982	15000	3.0	0.25
" "	E-705	1984	50000	20	3.0 (1500 hr run)
Malamud, Watts	E-557	1983	15000	10-35[3]	1-4[3]
" "	E-672	1985	25000	20[4]	3[4] (2x1000 hr runs)
Pless	E-636	1985	40000	15[5]	4[5] (3 mon. run, 1/4 of request)

Notes: 0) all numbers are rough estimates
1) beam time including interspill time for fixed target
2) units of Cyber-years even if other computers are to be used.
3) Larger number projected on basis of present MPS algorithms. Smaller number assumes possible improvement.
4) Assuming improved MPS algorithms
5) Assuming a 1 1/2 order of magnitude improvement over present algorithms. They expect even more improvement: "We've got to!"

Table II
Speakers at Advanced Computer Seminar Series

H.D. Kung	Carnegie Mellon	Systolic arrays
C. Maples	LBL	"Midas" multiproc. system
J. Schwartz	NYU	Ultra computer
J. Dennis	MIT	Data flow computers
W. Sippach	Columbia	Data driven system
D. Kuck	Illinois	Super computer compilers
B. Smith	Denelcor, Inc.	Design of the HEP
H. Walsch & F. Ris	IBM	Scientific computers
J. Abraham	Illinois	VSLI design
G. Fox	Cal Tech	Microprocessor arrays for physics

Table III

Comparison Criteria for ACP Processor Comparison Study

$ for CPU, 100 Qty, plus real estate
$ for 1 Mb memory, 100 Qty

Cycle time
Fixed point operation speed
Floating point operation speed
Physics bench mark

16 bits or 32 bits
Address space
Hardware memory plateaus - increments of memory
Virtual memory in hardware?

Existing Compilers?
- Ease of use
- Improvability

Upward compatibility of hardware

Ease of "coprocessor" interface
- calls
- memory transfer

Development costs
- Development systems, software
- Compiler

Table IV
List of Processors for ACP Comparison Study

*	CDC	Cyber 175
*	DEC	VAX 11-780 + array processor
*	SLAC	168E
*	SLAC/CERN	3081E
*	Motorola	68000 and descendents
*	Intel	8086 and descendents
*	National Semi	16032
*	HP	9000 32 bit chip set
*	DEC	μJ11
*	DEC	VAX chip
	μNova	
	TI	99000
	Zilog	Z8000
	Intel	IAPX 432
	Commodore	65000
	Bell Labs	Bellmac - 32
	IBM	?
	Various	Small computer boards

* Benchmark activity actively being pursued on these processors.

FIGURE CAPTIONS

1. Level 1 multiprocessor concept.
2. Level 2 (and 3) multiprocessor concepts.
3. Level 4 multiprocessor concept.
4. A list (or relation) shown schematically.
5. Schematic view of list development in a typical track reconstruction program.
6. Simple list operations (modification and relational algebra operations shown schematically.
7. Simple projection operation shown schematically with a typical example of its use.

Level 1:

Figure 1

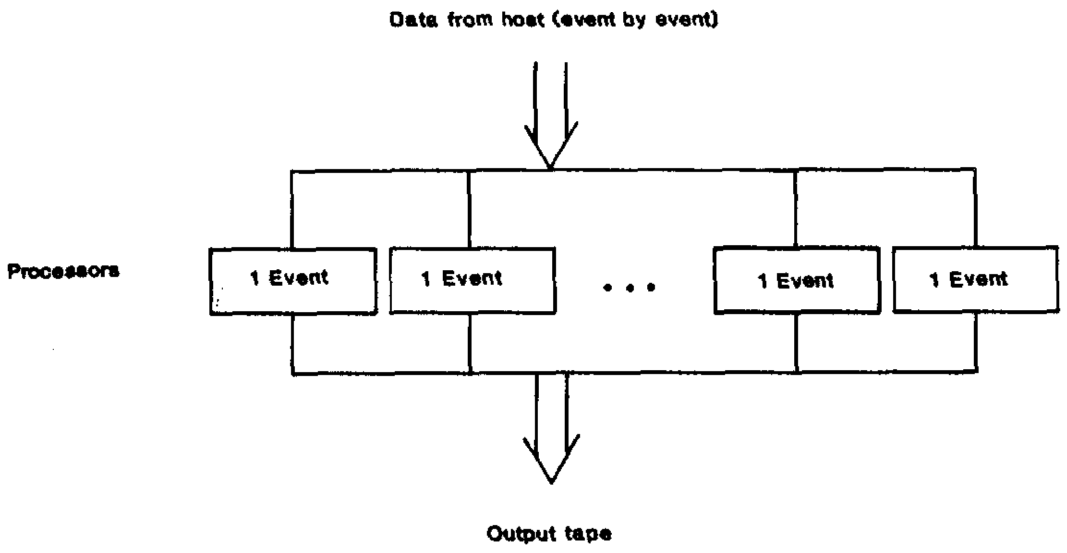
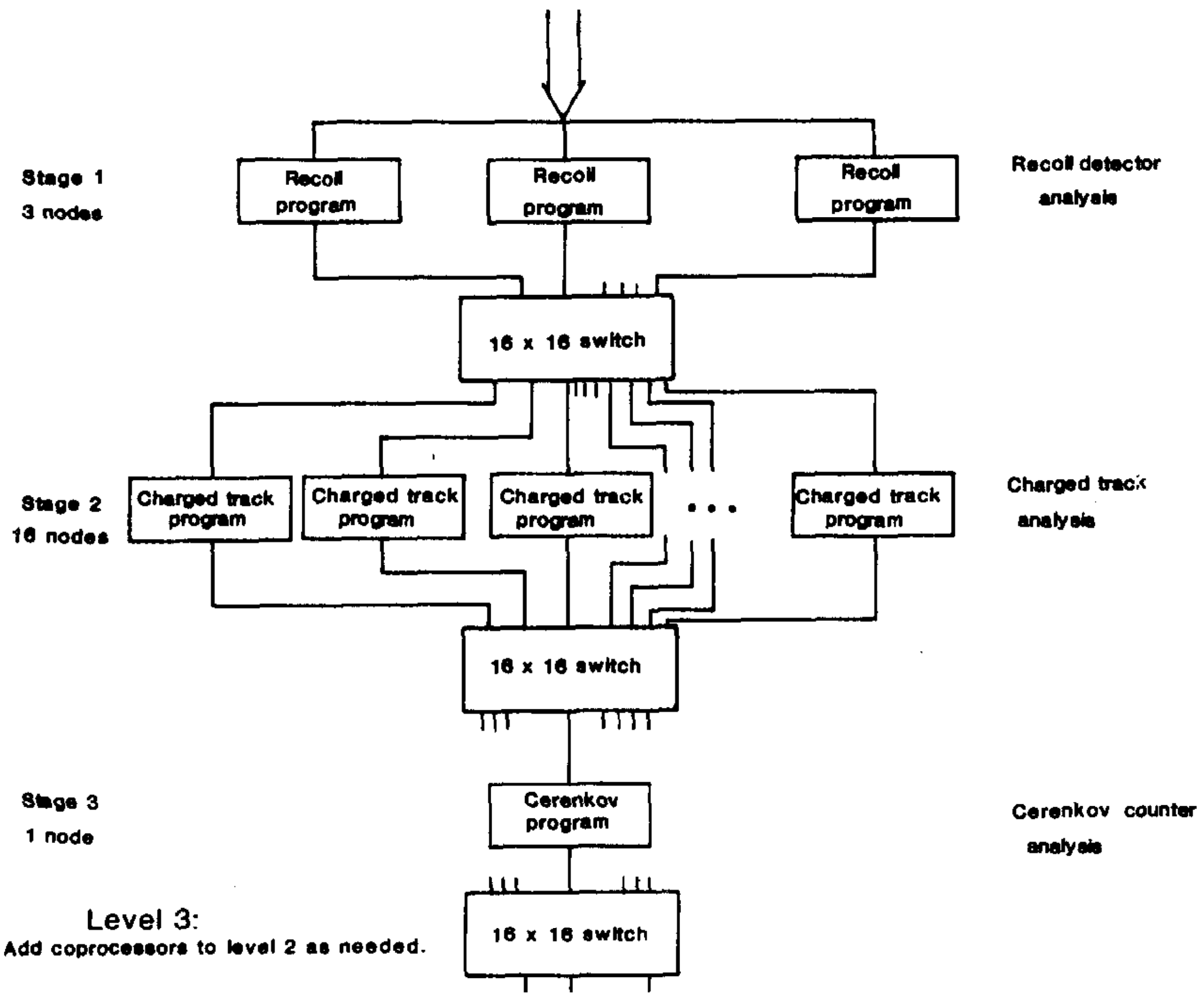


Figure 2

Level 2:



Level 3:
Add coprocessors to level 2 as needed.

Level 4:

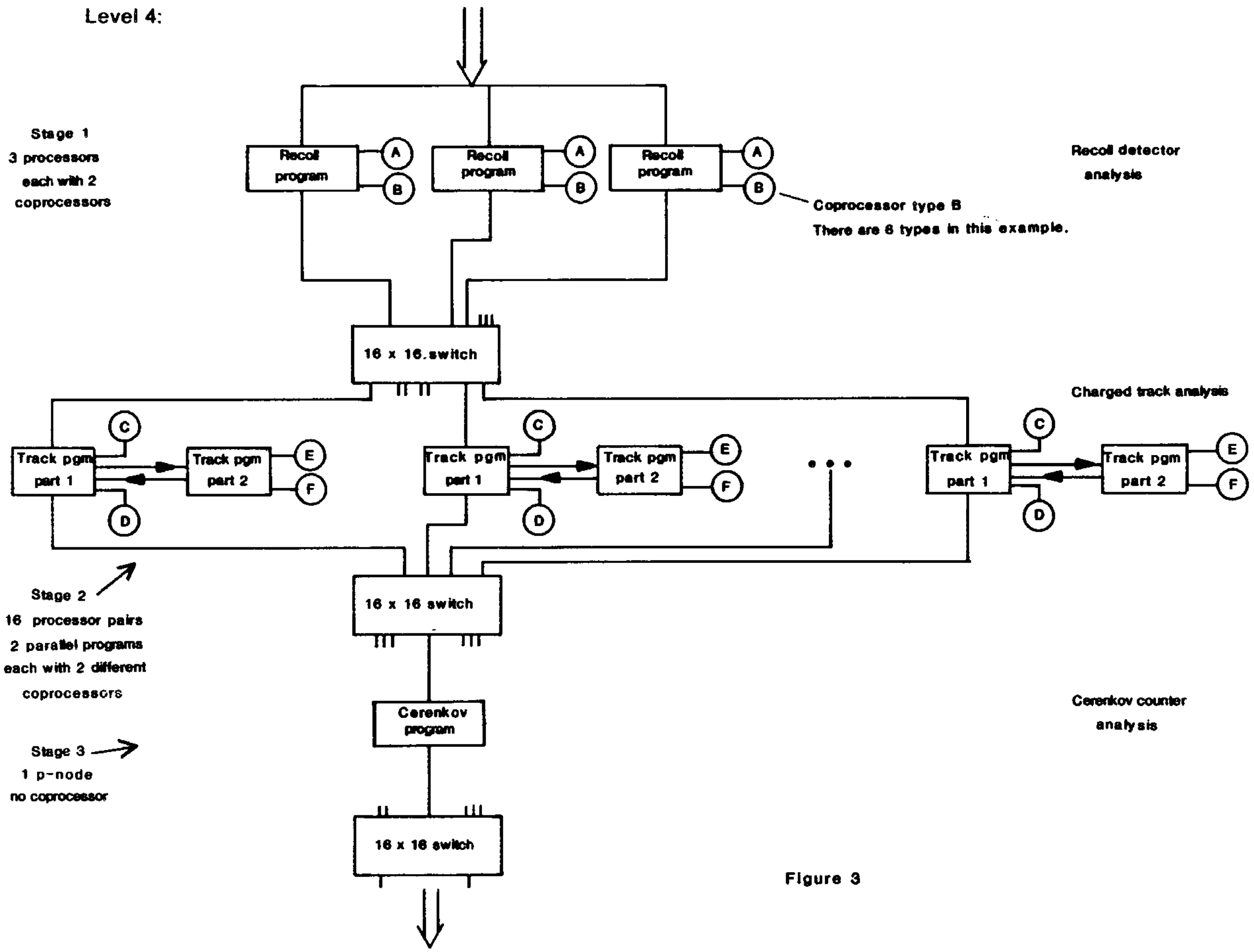
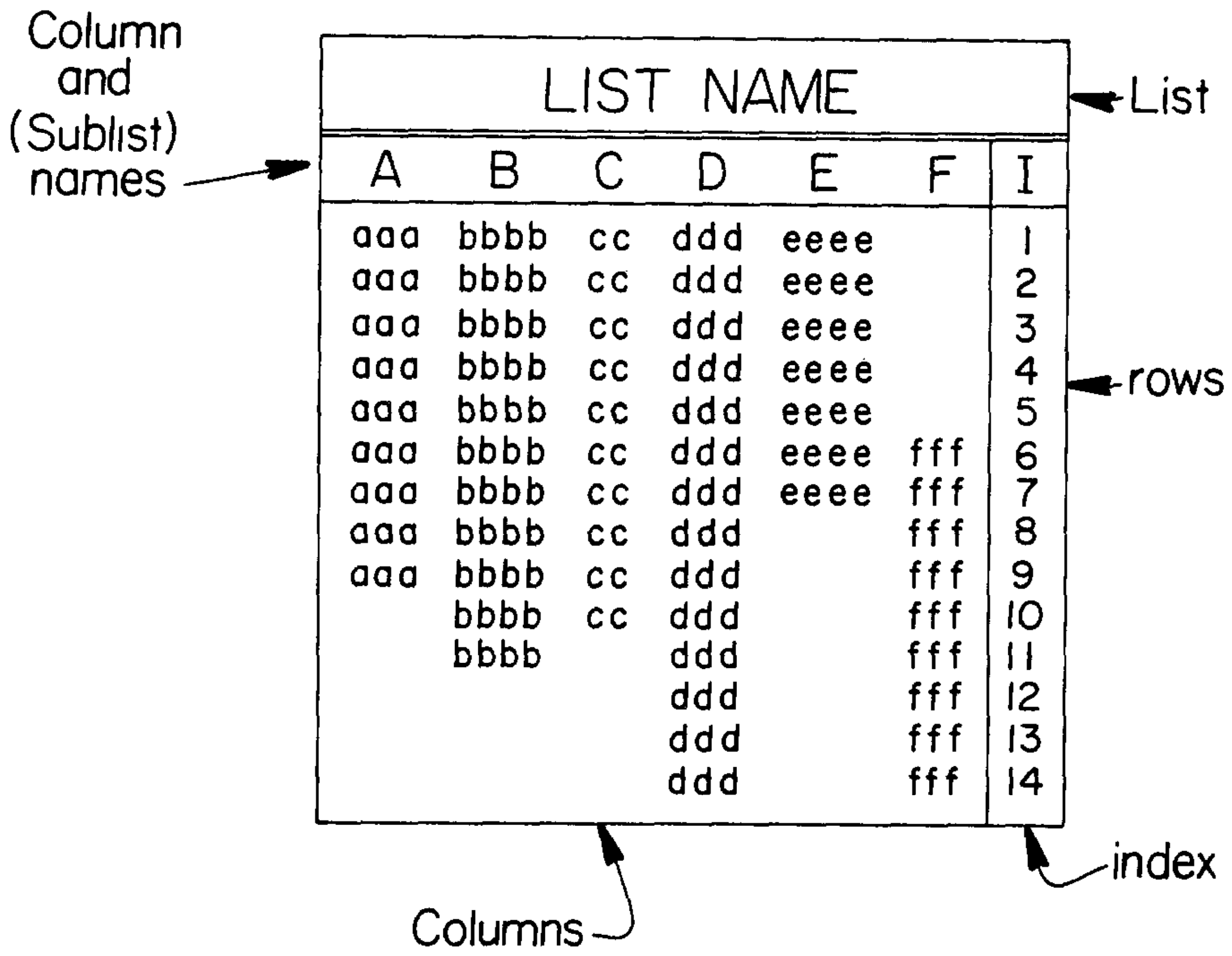
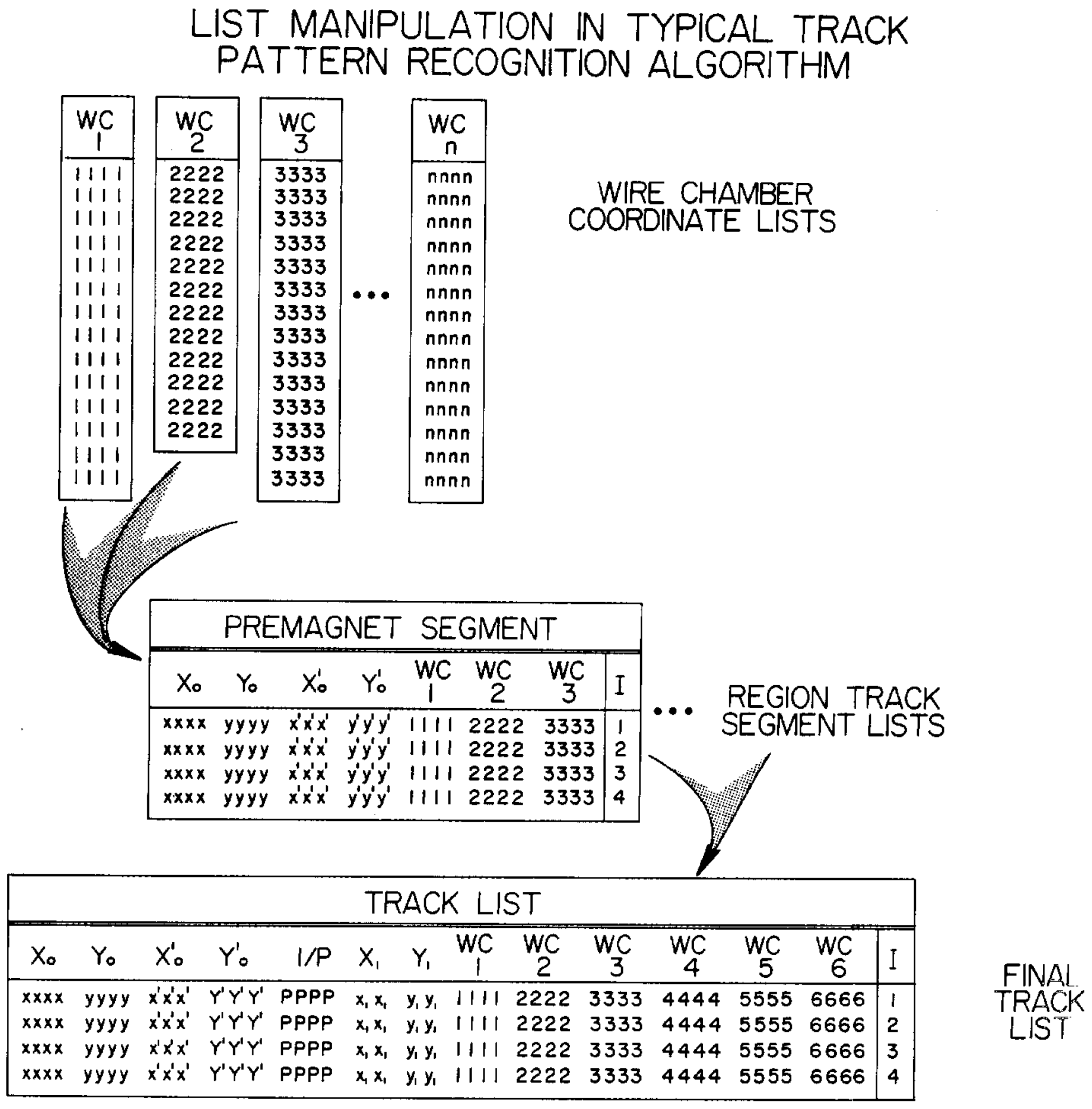


Figure 3



LIST NAME						
A	B	C	D	E	F	I
aaa	bbbb	cc	ddd	eeee		1
aaa	bbbb	cc	ddd	eeee		2
aaa	bbbb	cc	ddd	eeee		3
aaa	bbbb	cc	ddd	eeee		4
aaa	bbbb	cc	ddd	eeee		5
aaa	bbbb	cc	ddd	eeee	fff	6
aaa	bbbb	cc	ddd	eeee	fff	7
aaa	bbbb	cc	ddd		fff	8
aaa	bbbb	cc	ddd		fff	9
	bbbb	cc	ddd		fff	10
	bbbb		ddd		fff	11
			ddd		fff	12
			ddd		fff	13
			ddd		fff	14

Figure 4


LIST MANIPULATION IN TYPICAL TRACK PATTERN RECOGNITION ALGORITHM
WIRE CHAMBER COORDINATE LISTS
•••
REGION TRACK SEGMENT LISTS
FINAL TRACK LIST

WC 1	WC 2	WC 3	•••	WC n
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111	2222	3333		nnnn
1111		3333		nnnn
1111		3333		nnnn

PREMAGNET SEGMENT

X_0	Y_0	X'_0	Y'_0	WC 1	WC 2	WC 3	I
xxxx	yyyy	x'x'x'	y'y'y'	1111	2222	3333	1
xxxx	yyyy	x'x'x'	y'y'y'	1111	2222	3333	2
xxxx	yyyy	x'x'x'	y'y'y'	1111	2222	3333	3
xxxx	yyyy	x'x'x'	y'y'y'	1111	2222	3333	4

TRACK LIST

X_0	Y_0	X'_0	Y'_0	I/P	X_1	Y_1	WC 1	WC 2	WC 3	WC 4	WC 5	WC 6	I
xxxx	yyyy	x'x'x'	Y'Y'Y'	PPPP	$x_1 x_1$	$y_1 y_1$	1111	2222	3333	4444	5555	6666	1
xxxx	yyyy	x'x'x'	Y'Y'Y'	PPPP	$x_1 x_1$	$y_1 y_1$	1111	2222	3333	4444	5555	6666	2
xxxx	yyyy	x'x'x'	Y'Y'Y'	PPPP	$x_1 x_1$	$y_1 y_1$	1111	2222	3333	4444	5555	6666	3
xxxx	yyyy	x'x'x'	Y'Y'Y'	PPPP	$x_1 x_1$	$y_1 y_1$	1111	2222	3333	4444	5555	6666	4

Figure 5

SIMPLE LIST OPERATIONS USED IN RECONSTRUCTION ALGORITHMS

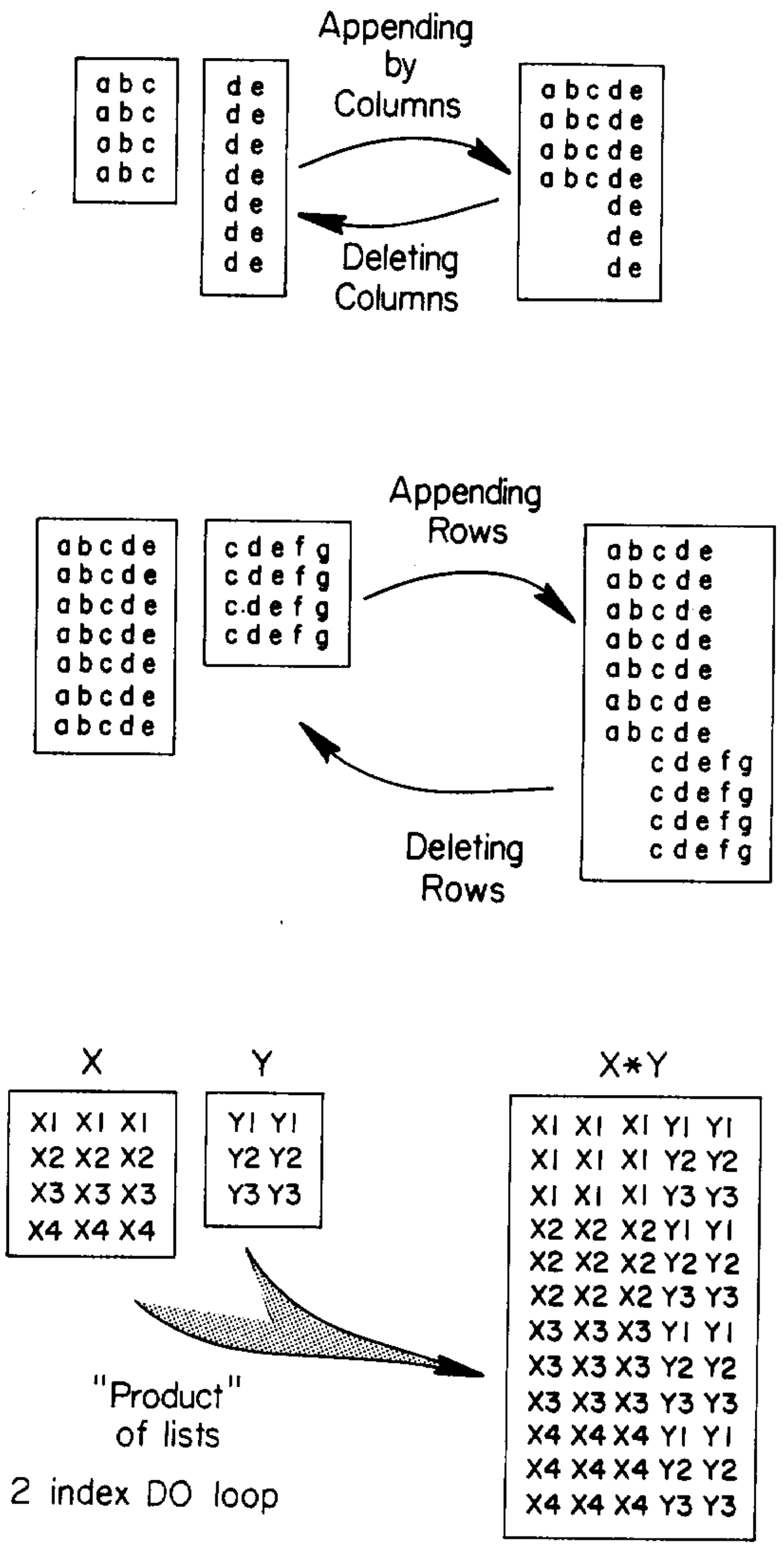


Figure 6

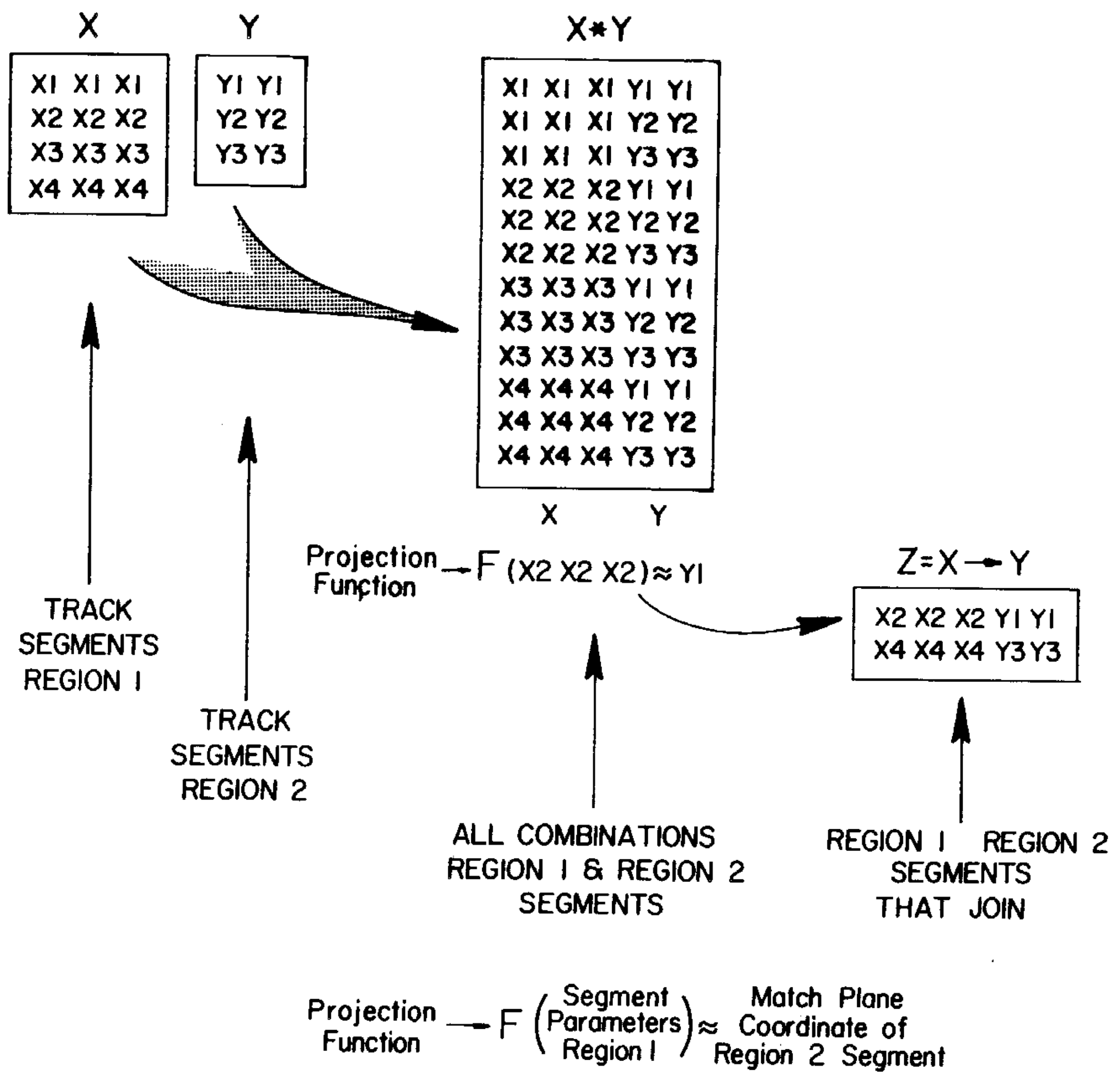

X
X1 X1 X1
X2 X2 X2
X3 X3 X3
X4 X4 X4
Y
Y1 Y1
Y2 Y2
Y3 Y3
X*Y
X1 X1 X1 Y1 Y1
X1 X1 X1 Y2 Y2
X1 X1 X1 Y3 Y3
X2 X2 X2 Y1 Y1
X2 X2 X2 Y2 Y2
X2 X2 X2 Y3 Y3
X3 X3 X3 Y1 Y1
X3 X3 X3 Y2 Y2
X3 X3 X3 Y3 Y3
X4 X4 X4 Y1 Y1
X4 X4 X4 Y2 Y2
X4 X4 X4 Y3 Y3
X
Y
Projection Function → F (X2 X2 X2) ≈ Y1
Z=X → Y
X2 X2 X2 Y1 Y1
X4 X4 X4 Y3 Y3
TRACK SEGMENTS REGION 1
TRACK SEGMENTS REGION 2
ALL COMBINATIONS REGION 1 & REGION 2 SEGMENTS
REGION 1 REGION 2 SEGMENTS THAT JOIN
Projection Function → F (Segment Parameters Region 1) ≈ Match Plane Coordinate of Region 2 Segment

Figure 7